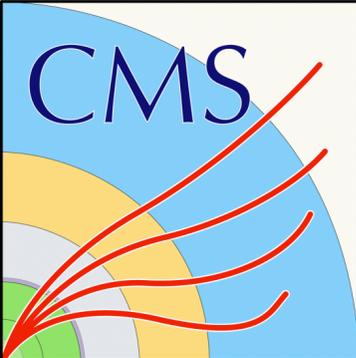


12345:

Lessons Learned building an Analysis Framework around RDataFrame and CMS NanoAOD

Nicholas Manganelli
PhD Candidate, Compact Muon Solenoid Collaboration

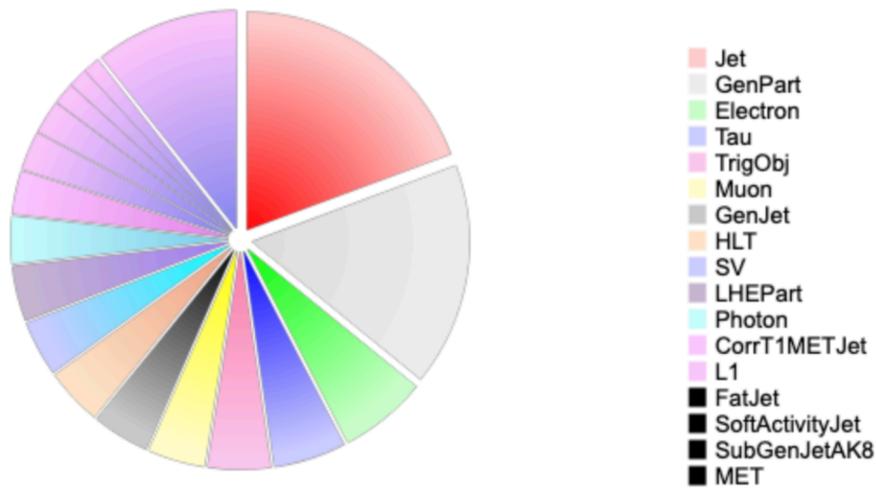


12345:

Lessons Learned building an Analysis Framework around RDataFrame and CMS NanoAOD

Nicholas Manganelli
PhD Candidate, Compact Muon Solenoid Collaboration

Thanks to the ROOT team!

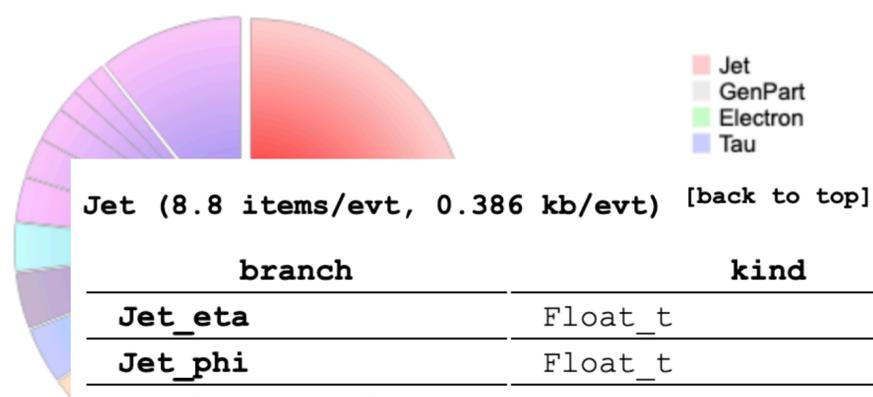


Compact Muon Solenoid's NanoAOD

- Data format for proton-proton collisions
 - Root-based, O(1kB) per event
- High Level (not all particles, hits, tracks, etc.)
- Data stored as scalars and **Jagged arrays** of **primitive** types (int, float, bool, ...)

Event data

collection	kind	vars	items/evt	kb/evt
Jet	collection	42	8.77	0.386
GenPart	collection	9	50.82	0.330
Electron	collection	63	1.15	0.130
Tau	collection	47	1.40	0.107
TrigObj	collection	11	9.88	0.093
Muon	collection	54	0.92	0.085
GenJet	collection	7	7.66	0.084
HLT	singleton	651	1.00	0.084
SV	collection	15	2.68	0.082
LHEPart	collection	9	8.45	0.081
Photon	collection	31	1.49	0.070
CorrT1METJet	collection	6	7.34	0.064
L1	singleton	321	1.00	0.056
FatJet	collection	50	0.36	0.049
SoftActivityJet	collection	4	5.96	0.031
SubGenJetAK8	collection	5	2.42	0.028
MET	singleton	12	1.00	0.024
SubJet	collection	17	0.49	0.019
GenJetAK8	collection	7	1.24	0.018
PuppiMET	singleton	7	1.00	0.016
IsoTrack	collection	15	0.34	0.013
LHE	singleton	11	1.00	0.012
PV	singleton	8	1.00	0.011



Compact Muon Solenoid's NanoAOD

- Data format for proton-proton collisions
 - Root-based, O(1kB) per event
- High Level (not all particles, hits, tracks, etc.)
- Data stored as scalars and **Jagged arrays of primitive** types (int, float, bool, ...)
- Contains “Collections” for muons, electrons, jets, generator-level particles, with cross-references via index-positions in linked collection
 - Coll. structure via naming convention
- Important collections: ~two-dozen variables

Even

Jet

Gen

Ele

Tau

Tri

Muo

Gen

HLT

SV

LHE

Pho

Cor

L1

Fat

Sof

Sub

MET

Sub

Gen

Pup

Iso

LHE

PV

branch	kind	b/event	b/item	plot	%
Jet_eta	Float_t	20.9	2.4	■	5.3%
Jet_phi	Float_t	20.5	2.3	■	5.2%
Jet_btagDeepFlavB	Float_t	17.3	2.0	■	4.4%
Jet_btagDeepFlavC	Float_t	16.7	1.9	■	4.2%
Jet_mass	Float_t	15.8	1.8	■	4.0%
Jet_pt	Float_t	15.7	1.8	■	4.0%
Jet_btagCSVV2	Float_t	14.7	1.7	■	3.7%
Jet_btagDeepB	Float_t	14.5	1.7	■	3.7%
Jet_btagDeepC	Float_t	14.2	1.6	■	3.6%
Jet_puIdDisc	Float_t	13.8	1.6	■	3.5%
Jet_qgl	Float_t	13.5	1.5	■	3.4%
Jet_cRegCorr	Float_t	13.3	1.5	■	3.4%
Jet_rawFactor	Float_t	13.3	1.5	■	3.4%
Jet_muonSubtrFactor	Float_t	13.2	1.5	■	3.4%
Jet_bRegCorr	Float_t	12.9	1.5	■	3.3%
Jet_btagCMVA	Float_t	12.6	1.4	■	3.2%
Jet_neEmEF	Float_t	11.9	1.4	■	3.0%
Jet_chFPV1EF	Float_t	11.8	1.3	■	3.0%
Jet_neHEF	Float_t	11.5	1.3	■	2.9%
Jet_chFPV0EF	Float_t	11.3	1.3	■	2.9%
Jet_chHEF	Float_t	11.0	1.3	■	2.8%
Jet_bRegRes	Float_t	10.6	1.2	■	2.7%
Jet_cRegRes	Float_t	10.5	1.2	■	2.7%
Jet_chFPV3EF	Float_t	10.4	1.2	■	2.6%
Jet_chFPV2EF	Float_t	10.1	1.2	■	2.6%
Jet_nConstituents	Int_t	7.5	0.9	■	1.9%
Jet_area	Float_t	6.8	0.8	■	1.7%
Jet_partonFlavour	Int_t	4.9	0.6	■	1.3%
Jet_genJetIdx	Int_t(index to Genjet)	4.1	0.5	■	1.0%
	singleton	11	1.00	0.012	
	singleton	8	1.00	0.011	

What did I do?

- Built an RDataFrame-based analysis framework
 - targeting a **specific** analysis, potentially for multiple related ‘decay channels’
 - Wanted something fast, NanoAOD-compatible, using **python** as the interface
 - Needed to be scalable - quick tests (fast turnaround to make decisions), scale up to processing billions of events with $O(100)$ systematic variations
 - Development partially overlapped bamboo (2019 - 2021 principally)
 - **SPOILERS**: Discovered how **not** to do a lot of things! <- good chunk of this presentation

“1 out of 5 - would not recommend”



Lesson 10:

Don't work alone

- solo-graduate student (all PostDocs/ GradStudents finished and left group early on)
- Few resources/knowledge of what other RDF users were doing (learned of bamboo ~6 months deep in dev)
- All the good documentation, examples, etc. are *very appreciated!*



credit

```

#Make sure the nominal is done first so that categorization is successful
for sysVarRaw, sysDict in sorted(sysVariations.items(), key=lambda x: "$NOMINAL" in x[0], reverse=True):
    #skip systematic variations on data, only do the nominal
    if isData and sysVarRaw != "$NOMINAL":
        continue
    #Only do systematics that are in the filter list (storing raw systematic names...
    if sysVarRaw not in sysFilter:
        continue
    #get final systematic name
    sysVar = sysVarRaw.replace("$NOMINAL", "nom").replace("$LEP_POSTFIX", sysDict.get('lep_postfix', '')).replace("$ERA", era)
    #skip making MET corrections unless it is: Nominal or a scale variation (i.e. JES up...)
    isWeightVariation = sysDict.get("weightVariation", False)
    #jetMask = sysDict.get("jet_mask").replace("$SYSTEMATIC", sysVar).replace("$LEP_POSTFIX", sysDict.get('lep_postfix', ''))
    #jetPt = sysDict.get("jet_pt_var")
    #jetMass = sysDict.get("jet_mass_var")
    #Name histograms with their actual systematic variation postfix, using the convention that HISTO_NAME__nom is
    # the nominal and HISTO_NAME__$SYSTEMATIC is the variation, like so:
    syspostfix = "__nom" if sysVarRaw == "$NOMINAL" else "__{}".format(sysVar)
    #Rename systematics on a per-sample basis, rest of code in the eraAndSampleName cycle
    systematicRemapping = sysDict.get("sampleRemapping", None)
    #name branches for filling with the nominal postfix if weight variations, and systematic postfix if scale variation (jes_up, etc.)
    branchpostfix = None
    if isWeightVariation:
        branchpostfix = "__nom"
    else:
        branchpostfix = "__" + sysVar
    leppostfix = sysDict.get("lep_postfix", "") #No variation on this yet, but just in case
    combineHistoVariables += [templateVar.format(bpf=branchpostfix) for templateVar in combineHistoTemplate]

fillJet = "FTAJet{bpf}".format(bpf=branchpostfix)
fillJetEnumerated = "FTAJet{{n}}{bpf}".format(bpf=branchpostfix)
fillJet_pt = "FTAJet{bpf}_pt".format(bpf=branchpostfix)
fillJet_phi = "FTAJet{bpf}_phi".format(bpf=branchpostfix)
fillJet_eta = "FTAJet{bpf}_eta".format(bpf=branchpostfix)

for dnode in defineNodes[eraAndSampleName][decayChannel][Hstart:Hstop]:
    model = dnode[0]
    defHName = model.fName
    if isData:
        variables = dnode[1:-1] # throw out the weight in the tuple definition
    else:
        variables = dnode[1:]
    #Need to determine which kind of histo function to use... have to be careful, this guess will be wrong if anyone ever does an unweighted
    if defHName in histoNodes[eraAndSampleName][decayChannel]:
        raise RuntimeError(f"This histogram name ({defHName}) already exists in memory or is intentionally being overwritten" \
            " {eraAndSampleName} - {decayChannel}")
    else:
        try:
            histoNodes[eraAndSampleName][decayChannel][defHName] = categoryNode.Histo3D(model, *variables)
        except Exception:
            for variable in variables:
                if variable not in listOfColumns:
                    print(f"variable {variable} is not available in the columns being histogrammed for {eraAndSampleName} - {decayChannel}")

```

```

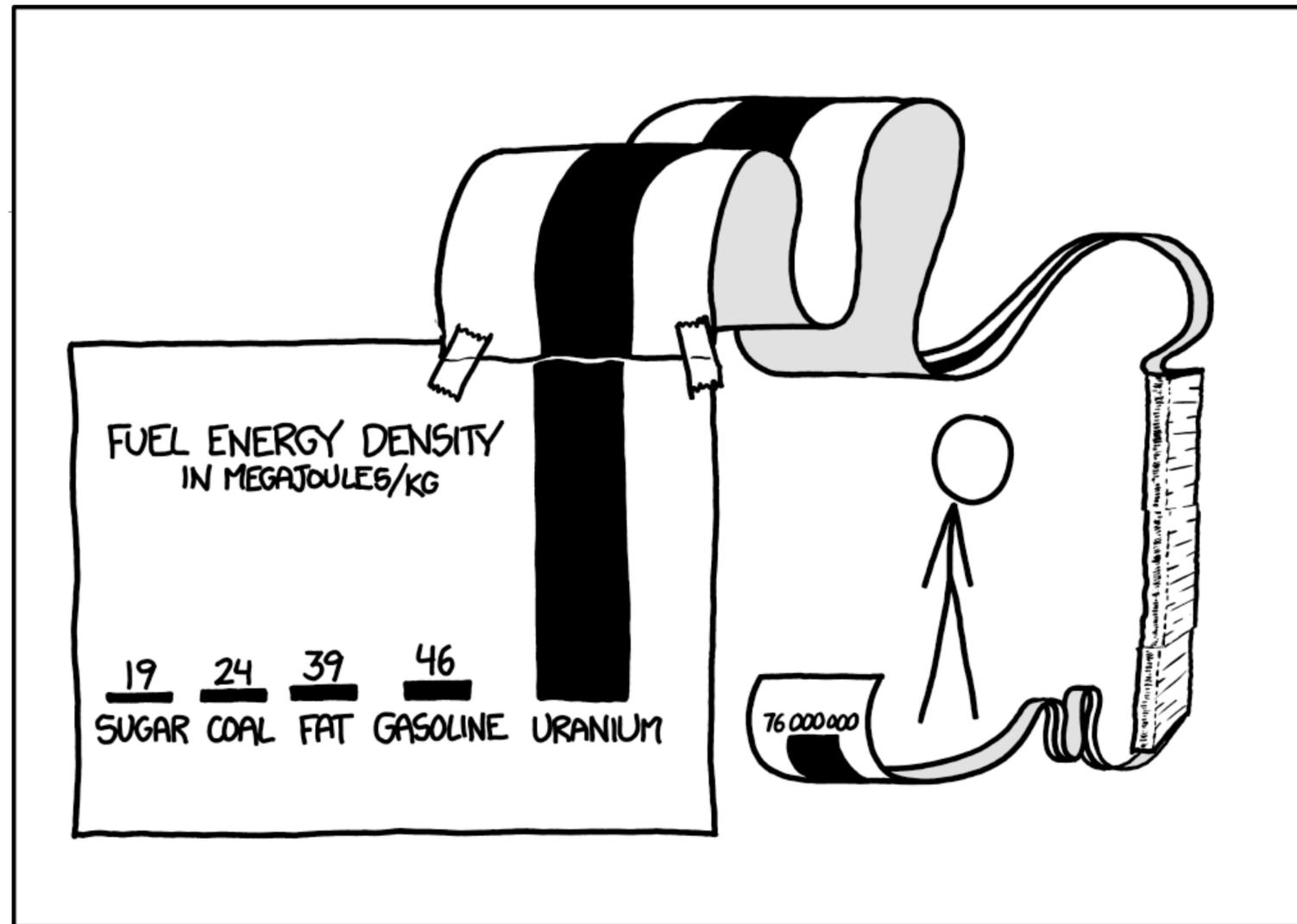
HT_Model = ROOT.RDF.TH3DModel(f"{eraAndProcessName}__{decayChannel}__{categoryName}__HT{histopostfix}",
    f"H_{{T}} ({{histopostfix[3:]}}); Jet Multiplicity; Medium b-Tagged Jet Multiplicity; H_{{T}}",
    len(nJetArr)-1, nJetArr, len(nBTagArr)-1, nBTagArr, len(HTArr)-1, HTArr)
defineNodes[eraAndSampleName][decayChannel].append(
    (HT_Model,
    f"n{fillJet}",
    f"nMedium{tagger}{branchpostfix}",
    f"HT{branchpostfix}",
    wgtVar)
)

```

Lesson 9:

- Running systematic variations in the same event loop
- Framework implementation predating Vary
- Manual book-keeping via string substitutions
- Weight-based: Histo1D(model, variableA_NOM, wgt_SYST_N)
- Non Weight-based: Histo1D(model, variableA_SYST_M, wgt_SYST_M)

Lesson 8: SCALING



SCIENCE TIP: LOG SCALES ARE FOR QUITTERS WHO CAN'T FIND ENOUGH PAPER TO MAKE THEIR POINT *PROPERLY*.

<https://xkcd.com/1162>

- I should have watched scaling behaviors
 - Early prototypes with systematics worked fine
 - Scaling up to the full set of ~75 systematic variations made memory use a concern! 
 - batch resources ~ 2GB/core
 - histograms copied per thread
- Good idea: backup plan for running either 1, *some, all* variations easily (keeping an eye on file-output/*tracking*/merging!) 
- Good idea: Atomic-storage ala boost::histogram (narf)

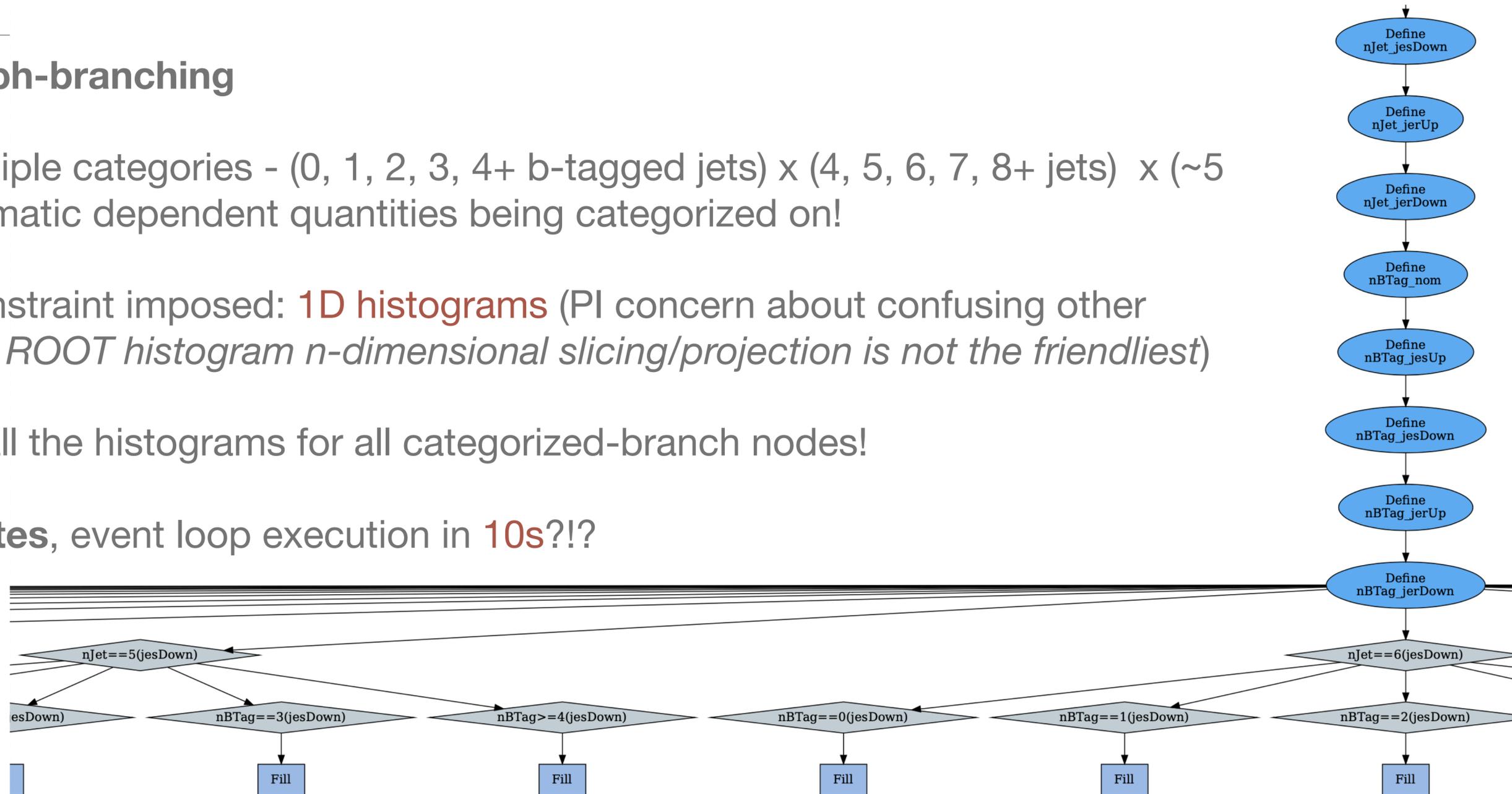
Lesson 8: SCALING



- **Categorization via graph-branching**

- Analysis required multiple categories - (0, 1, 2, 3, 4+ b-tagged jets) x (4, 5, 6, 7, 8+ jets) x (~5 systematics) — systematic dependent quantities being categorized on!
- Driven by external constraint imposed: **1D histograms** (PI concern about confusing other students - admittedly, *ROOT histogram n-dimensional slicing/projection is not the friendliest*)
- Very costly to define all the histograms for all categorized-branch nodes!
 - Setup time of **minutes**, event loop execution in **10s?!?**

- **Prefer ND Histos!**



Lesson 6: CONFIGURATION



Lesson 6: CONFIGURATION

Lesson 7: NON-UNIFORM SCALING DIMENSIONS



- **Non-uniform scaling dimensions** are a good way to trip up
- During analysis, we needed to take a few samples and **split them** into 2-4 sub-samples
 - Different weight scaling, naming, histogram grouping for statistical inference
 - Thought it would be great ('cute') to still run over the sample *once*... **failure to apply KISS**
 - So I introduced some *additional* splits in the computation graph node... a lot of nontrivial work and effort in order to do this for 'production-ready' case
 - Voilà, 4x the histograms... and ~4x the memory (**16GB** for most important background when using 1D histograms over many variables simultaneously for 8 threads!)
 - Also lead to discovery of a problem with [simultaneous Snapshots](#)

Lesson 6: CONFIGURATION

Steering code via configuration files is especially useful with JIT-ing via RDataFrame's python interface

Store quantities to Define or Filter-on in e.g. YAML* **

Round-trip bookkeeping without modifying everything by hand...

semi-arbitrary code execution (yes, use safe-load!)

* (Almost) Turing complete? Eh...

** Roundtrip read/update/write with comments is great, though

```
splitProcess:
  ID:
    nFTAGenJet/FTAGenHT: true
    subera: false
    unpackGenTtbarId: true
  processes:
    ttother_DL-GF_fr:
      filter: nAdditionalBJets < 2 && nFTAGenLep == 2 && nFTAGenJet >= 7 && FTAGenHT
        >= 500
      fractionalContribution: 0.894147882307
      effectiveCrossSection: 1.3911901673
      snapshotPriority: 2
      nEventsPositive: 8737826
      nEventsNegative: 44399
      sumWeights: 631625527.5556817
      sumWeights2: 46360359398.90896
      nominalXS: 1.4086498734234616
      nominalXS2: 2.3058633536195604e-07
      effectiveXS: 1.2439297561214113
      effectiveXS2: 1.7981220203253497e-07
      nLep2nJet7GenHT500-550-nominalXS: 0.17719474915135627
      nLep2nJet7pGenHT500p-nominalXS: 1.4086498734234616
      nLep1nJet9GenHT500-550-nominalXS: 0.0
      nLep1nJet9pGenHT500p-nominalXS: 0.0
      nLep2nJet7GenHT500-550-effectiveXS: 0.156474525897916
      nLep2nJet7pGenHT500p-effectiveXS: 1.2439297561214113
      nLep1nJet9GenHT500-550-effectiveXS: 0.0
      nLep1nJet9pGenHT500p-effectiveXS: 0.0
    ttbb_DL-GF_fr:
      filter: nAdditionalBJets >= 2 && nFTAGenLep == 2 && nFTAGenJet >= 7 && FTAGenHT
        >= 500
      fractionalContribution: 0.894231918352
      effectiveCrossSection: 0.0659588897972
      snapshotPriority: 4
      nEventsPositive: 300696
      nEventsNegative: 1393
```

Computed and
round-trip'd
to the YAML

\$NOMINAL:

```
jet_mask: jet_mask
lep_postfix: ''
jet_pt_var: Jet_pt_nom
jet_mass_var: Jet_mass_nom
met_pt_var: METFixEE2017_T1_pt
met_phi_var: METFixEE2017_T1_phi
```

Indicates branch names of inputs for this variation

```
btagSF:
  CSVv2: Jet_btagSF_csvv2_shape
  DeepCSV: Jet_btagSF_deepcsv_shape
  DeepJet: Jet_btagSF_deepjet_shape
```

weightVariation: false

Variation type indicators for handling in the analyzer

isNominal: true

systematicSet: Flag for that backup plan to limit number of systematics in a run

- nominal

commonWeights:

```
pwgt_LSF__nom: "(FTALepton$LEP_POSTFIX_SF_nom.size() > 1 ? FTALepton$LEP_POSTFIX_SF_nom.at(0) * FTALepton$LEP_POSTFIX_SF_nom.at(1) : FTALepton$LEP_POSTFIX_
EGamma_HLT_ZVtx_SF_nom: "return 0.991;"
EGamma_HLT_ZVtx_SF_unc: "return 0.001;"
pwgt_Z_vtx__nom: "return (abs(FTALepton$LEP_POSTFIX_pdgId.at(0, 0)) == 11 || abs(FTALepton$LEP_POSTFIX_pdgId.at(1, 0)) == 11 ) ? EGamma_HLT_ZVtx_SF_nom : 1
top_parton_pt_nnlo_nlo: "GenPart_pt[abs(GenPart_pdgId) == 6 && (GenPart_statusFlags & 8192) > 0]"
top_parton_sf_nnlo_nlo: "sqrt(0.103*exp(-0.0118 * top_parton_pt_nnlo_nlo) - 0.000134 * top_parton_pt_nnlo_nlo + 0.973)"
pwgt_top_pT_nnlo_nlo: "top_parton_sf_nnlo_nlo.at(0, 1.0) * top_parton_sf_nnlo_nlo.at(1, 1.0)"
prewgt__nom: "pwgt__LumiXS * puWeight * L1PreFiringWeight_Nom * pwgt_LSF__nom * pwgt_Z_vtx__nom * pwgt_ttbar_njet_multiplicity__$SYSTEMATIC * pwgt_top_
pwgt_btagSF_common: "pwgt__LumiXS * puWeight * L1PreFiringWeight_Nom * pwgt_LSF__nom * pwgt_Z_vtx__nom * pwgt_ttbar_njet_multiplicity__$SYSTEMATIC * pwg
```

finalWeights:

```
wgt__$SYSTEMATIC: "prewgt__$SYSTEMATIC * pwgt_btag__$SYSTEMATIC"
```

OSDL_\$ERA_jesTotalUp:

```
jet_mask: jet_mask__$SYSTEMATIC
lep_postfix: ''
jet_pt_var: Jet_pt_jesTotalUp
jet_mass_var: Jet_mass_jesTotalUp
met_pt_var: METFixEE2017_T1_pt_jesTotalUp
met_phi_var: METFixEE2017_T1_phi_jesTotalUp
```

JIT'd Definitions and weights for final histogram filling
Flexibility > Runtime execution

Lesson 5: All-In-One BOOKKEEPING

All-in-one meta-info storage is a double-edged sword

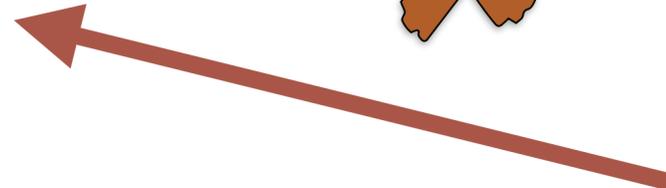
- Easier to look at information coherently
- More work to parse and compare simple information (i.e. just number of events between samples)
- In the end, a win IMO

```
- ELEM_D:
  era: '2018'
  subera: D
  isData: true
  nEvents: 56233597
  channel: ELEM
  channels:
  - ELEM
  source:
    NANOV6: dbs:/EGamma/Run2018D-Nano250ct2019-v1/NANOAOB
    NANOV7: dbs:/EGamma/Run2018D-02Apr2020-v1/NANOAOB
    NANOV7_CorrNov: list:/eos/user/n/nmangane/analysis/LongTermFilelists/2018_NANOV7_CorrNov__ELEM_D.txt
    NANOV7_CorrNov__ELEM: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELEM_D/ELEM/*Skim.root
    NANOV7_CorrNov__ELMu: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELEM_D/ELMu/*Skim.root
    NANOV7_CorrNov__MuMu: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELEM_D/MuMu/*Skim.root
    NANOV7_CorrNov_brokenpublishing: dbs:/EGamma/nmangane-NoveCampaign-bbf89cdf76a3c3d36b22aa33ba75d46e/USER instance=prod/phys03
- ELMu_A:
  era: '2018'
  subera: A
  isData: true
  nEvents: 4453465
  channel: ELMu
  channels:
  - ELMu
  source:
    NANOV6: dbs:/MuonEG/Run2018A-Nano250ct2019-v1/NANOAOB
    NANOV7: dbs:/MuonEG/Run2018A-02Apr2020-v1/NANOAOB
    NANOV7_CorrNov: list:/eos/user/n/nmangane/analysis/LongTermFilelists/2018_NANOV7_CorrNov__ELMu_A.txt
    NANOV7_CorrNov__ELEM: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELMu_A/ELEM/*Skim.root
    NANOV7_CorrNov__ELMu: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELMu_A/ELMu/*Skim.root
    NANOV7_CorrNov__MuMu: glob:/eos/user/n/nmangane/files/NANOV7_CorrNov/skims/2018/ELMu_A/MuMu/*Skim.root
    NANOV7_CorrNov_brokenpublishing: dbs:/MuonEG/nmangane-NoveCampaign-bbf89cdf76a3c3d36b22aa33ba75d46e/USER instance=prod/phys03
```

Lesson 4: STRUCTURE

```
Define("MyJet_mask", "Jet_pt > 30 && abs(Jet_eta) < 2.5...")
Define("MyJet_pt", "Jet_pt[MyJet_mask]")
Define("MyJet_phi", "Jet_phi[MyJet_mask]")
Define("MyJet_eta", ...)
...
Define("MyMuon_pt", "Muon_pt")
```

```
if sort_column:
    if not sort_ascending:
        events = events.Define(
            f"{output_collection}jettake",
            f"return Reverse(Argsort({input_collection}{sort_column}[{output_collection}jetmask]));",
        )
    else:
        events = events.Define(
            "jettake",
            f"return ArgSort({input_collection}{sort_column}[{output_collection}jetmask]);",
        )
else:
    events = events.Define(
        f"{output_collection}jettake",
        f"return {input_collection}idx[{output_collection}jetmask];",
    )
events = events.Define(
    f"n{output_collection[:-1]}", f"return Sum({output_collection}jetmask);"
)
for scol in sel_columns:
    # This can be replaced by .Select(Jet_*, ...) when that feature is supported
    events = events.Define(
        output_collection + scol,
        f"return Take({input_collection}{scol}[{output_collection}jetmask], {output_collection}jettake);",
    )
return events
```



- Analysis Framework retained for too long array-at-a-time manipulation (see left)
- MoA (Mess of Arrays), AoS, proxies?
- gist: structs versus arrays - lose a lot of RVec convenience
- python proxies (bamboo)
- RDataFrame-native proxy/object-like view (What's possible? how much could/should be Experiment responsibility? e.g. Schema - NanoAOD has ~1 new version per year)

sketch of select jets.py

Lesson 3: NumPy-LIKE



- Hard to adjust to numpy-like array manipulation
- How to create two *distinct* sub-collections with **complex multidimensional-cuts**?

```
Define("iso_mu_mask",  
      "Muon_pt > 30 && abs(Muon_eta) <= 2.4 &&  
      Muon_mediumId == true && Muon_pfIsoId >= 4")
```

```
Define("jpsi_cand_mu_mask",  
      "Muon_pt > 3 && abs(Muon_eta) <= 2.4 &&  
      Muon_looseId == true")
```

Simple Example: Create two sub-collections of muons, which **should not overlap**, but have different pT, isolation, and Id requirements...

Lesson 3: NumPy-LIKE



- Hard to adjust to numpy-like array manipulation
- How to create two *distinct* sub-collections with **complex multidimensional-cuts**?

```
Define("iso_mu_mask",  
      "Muon_pt > 30 && abs(Muon_eta) <= 2.4 &&  
      Muon_mediumId == true && Muon_pfIsoId >= 4")
```

```
Define("jpsi_cand_mu_mask",  
      "Muon_pt > 3 && abs(Muon_eta) <= 2.4 &&  
      Muon_looseId == true &&  
      iso_mu_mask == false")
```

Explicitly via boolean mask inversion!

Also applicable to awkward-array!

Essential, extremely useful!

Lesson 2: C++ Python

```
ROOT.gROOT.ProcessLine(".L SomeFunctions.cpp")
```

- Trial By Fire to learn
 - ROOT.gROOT.ProcessLine, ROOT.gInterpreter.Declare, hasattr, getattr, isinstance
- Progress Bars, Look Up Tables (e.g. TH2)
- Testing C++ instantiated objects working correctly

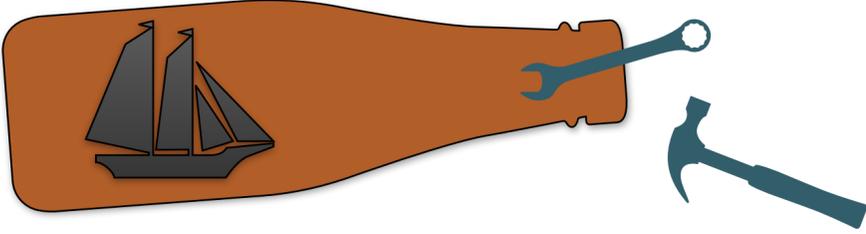
```
if type(lookupMap) == str:  
    #It's a string name, see if it's been declared in the ROOT instance  
    try:  
        if str(type(getattr(ROOT, lookupMap))) == "<class 'ROOT.map<string,vector<TH2Lookup*> >'>":  
            pass  
    except:  
        ROOT.gInterpreter.Declare("std::map<std::string, std::vector<TH2Lookup*>> {0};".format(lookupMap))  
    iLUM = getattr(ROOT, lookupMap)
```

Lesson 1: DE--ing and Code-building Code

- **pdb** (`pdb.set_trace()`) is essential when working from python
 - Can't always separate out simple test-scenario -> good way to dive into middle of code exec.
- C++ `cout` for event-loop checks
 - More than once I forgot to prevent that code's execution being skipped!



Lesson 1: DE--ing and Code-building Code

- **pdb** (`pdb.set_trace()`) is essential when working from python
 - Can't always separate out simple test-scenario -> good way to dive into middle of code exec.
- C++ `cout` for event-loop checks
 - More than once I forgot to prevent that code's execution being skipped! 
- **Ship-in-a-Bottle** Syndrome  biggest time-sink of analysis dev
 - Building C++ code inside python-formatted strings -> Not ideal, but not always preferable to move to pure C++
 - No easy/direct access to the the 'environment' *inside* the event loop -> "BreakPoint" Proposal?

“I never learned from a man who agreed with me.”

–Robert A. Heinlein

ROOT and the !ROOT Ecosystems

- Benchmarking the code and coming out fastest is fantastic
 - Factor 3x* is small compared to the O(1000)-O(10000) improvement RDF/coffea have against TTree::Draw-based frameworks (I know of several)
- Benchmarking the *user experience* is harder, but for many analysts, this time probably **dominates!**
 - Emphasis on new features like Vary, **collection-aggregates/object-like proxies**, DefinePerSample, SampleInfo, *Distributed*, etc. is the right way to go at this stage
 - As much **UX-enhancements** as performance-enhancements

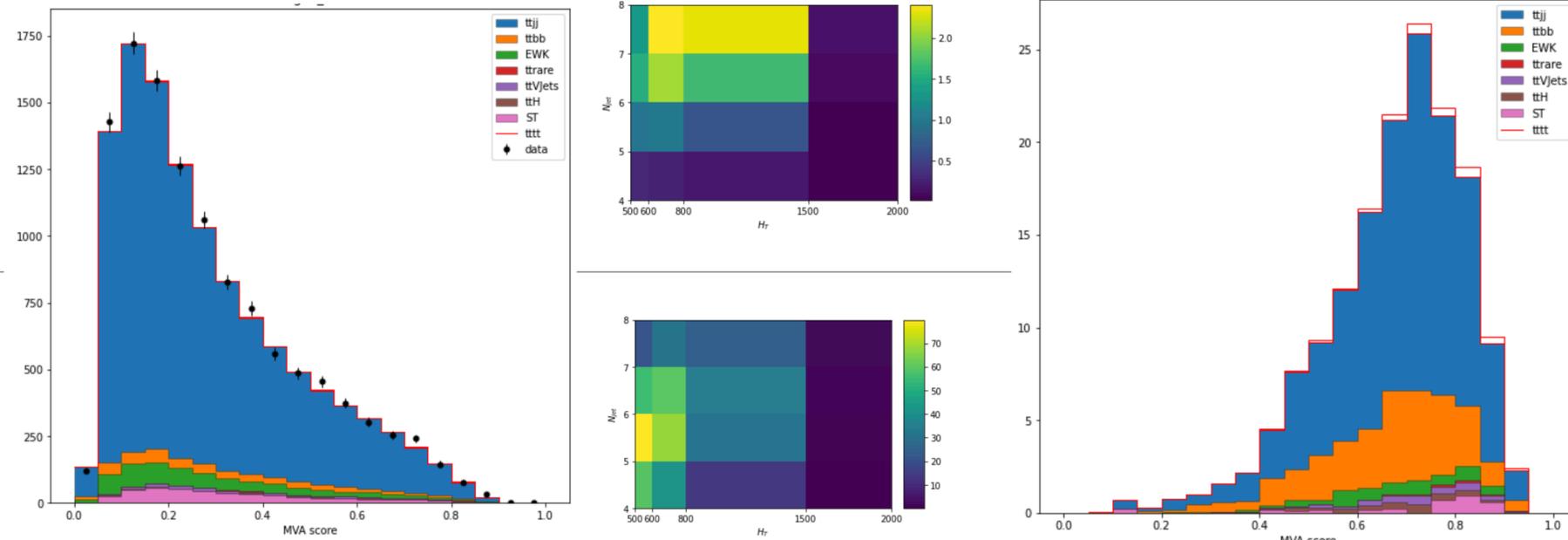


Coming soon

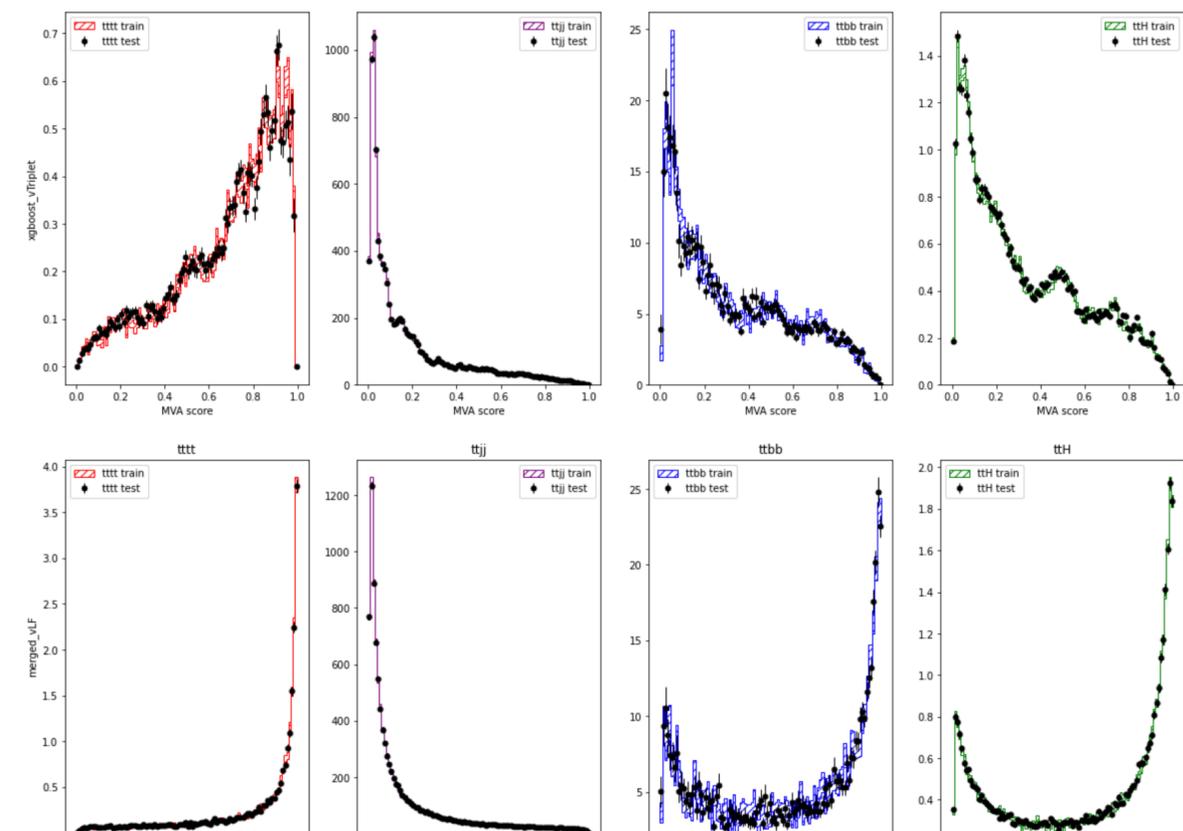
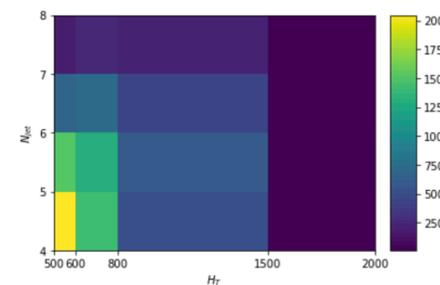
- Performance improvements (e.g. bulk processing, [ROOT PoW 2022](#))
- Collection aggregations ($\text{muon}_{\{pt,\eta,\phi\}}$ → muons) [being discussed](#)
- Simpler Pythonic interfaces (less C++ strings in Python code), [PoW 2022](#)
- Allow default values for missing branches, [PoW 2022](#), [GitHub issue](#)
- Debug symbols in jitted code (better error messages), [PoW 2022](#), [GitHub PR](#)
- Dataset specification with user-defined sample labels
- ...and more, see our [GitHub issue tracker](#)

Multi-Dimensional Histograms

- Easy, composable N-Dim histograms ($N = 5, 9, \dots$) ala boost::histogram are really great (ROOT7 convergent evolution? Categorical, circular, ...)
- Combination with [UHI](#) (standards) is extremely-powerful, game-changing...
 - I've been converting my workflows to this
 - Doubtful I'll ever look back...
- ROOT: `histo.GetYAxis().SetRange(yBinLow, yBinHigh); histo.GetZAxis().SetRangeUser(... ; ... histo.ProjectY(...)`
- `h[{"mva": "merged_vTriplet", "HT": hist.tag.Slicer()[700j : 1500j : sum], "nbttag": hist.tag.Slicer()[3j : 4j : sum}].project("dataset", "mva").plot1d(overlay="dataset")`



CMS Work In Progress (APS2022)



All plots from one single histogram (9D)

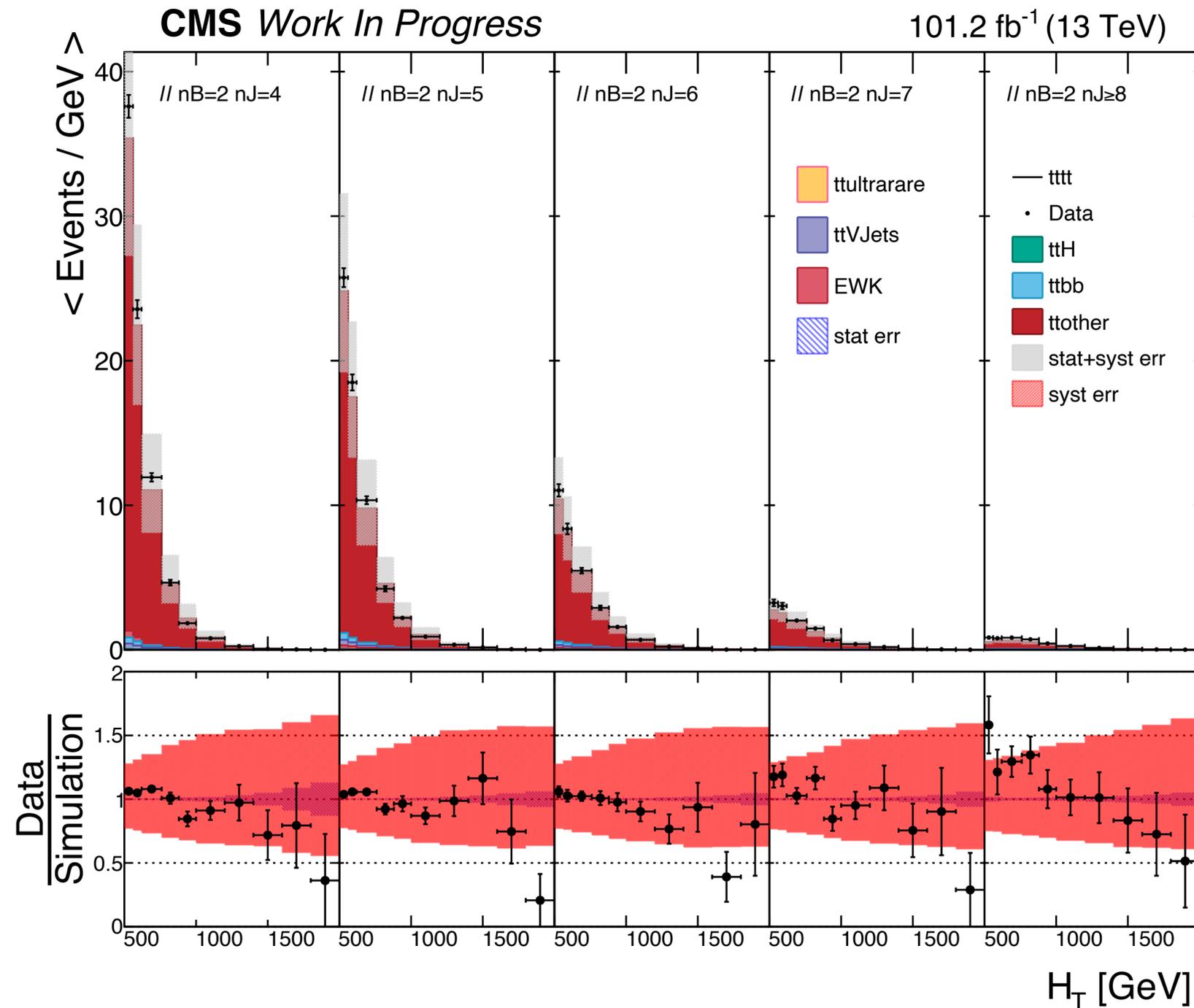
some dozen lines: slicing + plotting

BACKUP

Distributed and Scaling

- Potential with Distributed + Vary + DefinePerSample/SampleInfo to have a single call that executed an entire analysis processing (all variables x cuts/categories x systematic variations x samples...)
- What happens if this produces larger-than-memory results on the user?
 - Writing histograms to disk in efficient way from the worker-node?
 - Writing NumpyArrays out?

H_T (2 b-tagged jets)



Plotting

- It takes **74 * 50** histograms to make one PANEL in this plot (x 5 panels/canvas x 5 btag categories x 3 channels x 2 years x dozens of variables)
- ROOT's memory management made this painful early on, with Projections and Rebinning galore (not shown here!)
- Frustrating text scaling in different-sized pads (-> font precision 43 had its own quirks)
- Ticks, labelling them when margins -> 0
- aggregating and rebinning from python -> expensive for loops -> Multi-Dim Histograms preferable!

ML-training

- SOFIE might greatly simplify the issue of inference
- For training, being able to use the plethora of expertise, examples, code being developed by the huge data science and AI research communities would be really welcome -> “Generator” interface?